RB-002

Web ブラウザからの GPGPU を実現するプラグインとその応用
GPGPU Plug-in for the Web Browser

富澤 勇介† 高井 昌彰††
Yusuke Tomisawa Yoshiaki Takai

1. はじめに

膨大な処理量を要するリアルタイム 3D グラフィックスの専用演算装置であった GPU の高い処理性能に着目し，GPU をより汎用的な目的で利用しようとする GPGPU に注目が集まっている．GPU は，汎用演算装置として広く用いられている CPU と比べ，特に浮動小数点演算において数倍から数十倍の高い演算能力を有している．この高い演算能力を活用し，動画エンコードなど個人向けアプリケーションでの利用から，多数の GPU を束ねたハイパフォーマンスコンピューティング領域での利用まで，様々な分野やスケールでの利用が進められている．

一方，Web の領域おいても，GPU を活用しようという動きが広がっているが，現在提唱されている技術のいずれも，動画再生支援や 3D グラフィックスの提供など特定用途向けであり，汎用目的での利用を想定していない．

そこで本研究では，CPU や GPU などの様々な演算処理装置を統一的に取り扱うためのフレームワークである OpenCL を，ブラウザから JavaScript で取り扱えるようにするブラウザプラグインを新たに開発し，ウェブブラウザから GPGPU を利用できる基盤を構築する．

2. GPGPU と OpenCL

2.1 GPGPU

GPU は，グラフィックス処理，特にリアルタイム 3D グラフィックス処理を担う専用演算装置として，グラフィックボードやビデオカードなどのパッケージで広く利用されている．一般に，高精細な 3D グラフィックスには膨大な計算量が要求される．GPU は，汎用的な処理を行う CPU とは異なり，グラフィックス処理に特化した高性能な専用回路として，CPU に代わりこの処理要求に応えるものである．

グラフィックス処理専用であった GPU であるが，プログラマブルシェーダアーキテクチャの登場により，GPU の汎用目的での利用の可能性が見出されるようになった．これが，GPGPU (General-purpose computing on GPU) である．近年では，CUDA のような GPGPU 向けの高級言語による開発環境が登場しアプリケーションの構築が容易になったこともあり，広く普及し始めている．

†北海道大学大学院情報科学研究科, Graduate School of Information Science and Technology, Hokkaido University

††北海道大学情報基盤センター, Information Initiative Center, Hokkaido University

2.2 OpenCL

CUDA のように，高級言語による GPU プログラムの記述が可能な環境が整ってきたことで，GPU コンピューティングをより利用しやすくなってきた．しかし，多くの開発環境・言語は特定の GPU ベンダ向けになっており，複数のベンダの GPU に対応しようとすると，各 GPU に対応した環境や言語の習得が必要であり，開発・研究者にとって負担である．

OpenCL (Open Computing Language) は，こういった GPU ベンダを気にすることなく，様々なデバイスを統一的に扱えるようにするための汎用フレームワークである．OpenCL では，GPU だけでなく CPU や DSP などの演算装置を対象としており，様々なデバイスを混在させた異種混合演算環境を取り扱うことを想定している．

3. ウェブアプリケーションと OpenCL

3.1 ウェブアプリケーションの発展

Web に関する技術は，既存の技術では困難であった多様なメディア表現を可能にするように発展を続けてきた．単純に静的ページを閲覧するだけであったものが，ユーザの対話的操作やデータベースに蓄積した情報をもとにした動的ページ生成が行われるようになり，現在では，JavaScript による非遷移ページ操作や非同期通信，Adobe® Flash®などによるマルチメディアコンテンツ利用が広く普及している．

ウェブサイトを閲覧するためのソフトウェアに過ぎなかったウェブブラウザも，ウェブアプリケーションのプラットフォームとしての役割を担うようになってきた．

新しい技術は現在もなお提唱されており，アプリケーションプラットフォームとしてのウェブブラウザの役割はますます大きくなってきている．

3.2 ウェブにおける GPU 利用

WebGL は，ブラウザ上の JavaScript から HTML5 で規定されている canvas 要素内でリアルタイム 3D グラフィックスの描画を行えるようにするためのグラフィックス API である．また，動画共有などのようなマルチメディアコンテンツで広く用いられている Adobe® Flash®では，最新のリリースにおいて，GPU による動画再生支援機能を提供している．これにより，高解像度動画で要求されるような膨大な計算処理要求を GPU に担わせることができ，CPU にかかる負担を大きく減らすことができる．

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.

3.3　本システムの位置づけ

本システムでは，OpenCL をウェブブラウザ上で動作する JavaScript から呼び出せるようにすることで，Web アプリケーションにおける，GPGPU を含めた異種混合演算処理環境利用の基盤を構築する．

本来インタプリタ言語である JavaScript の処理速度はコンパイル方式のプログラムに比べると大きく劣る．また，CPU 上でシングルスレッドモデルとして実行される JavaScript では，GPU の資源が利用されることはなく，CPU においてもマルチコアプロセッサでは 1 つのコアしか用いられない．

本システムを利用すると，CPU や GPU などの利用可能な演算デバイスはすべて OpenCL 制御下に置かれ，アプリケーションは本システムを経由して OpenCL を呼び出すことで，様々なデバイスを利用できる．また，各デバイスでは，OpenCL C 言語で記述されビルドされたネイティブコードが実行されるため，実行パフォーマンスについても JavaScript のみによる処理と比べて大きく向上することが期待できる．さらに，OpenCL では並列計算モデルを採用しているため，近年の CPU や GPU のような並列あるいは超並列演算装置も有効に活用できる．

4.　プラグインの設計と実装

4.1　プラグインの概要

一般的な OpenCL の利用では，C や C++などのような各種プログラミング言語から OpenCL ランタイム API を呼び出すことで，OpenCL ランタイムが制御下に置くデバイスを利用する．一方，ウェブブラウザ上の JavaScript から OpenCL を利用しようとすると，JavaScript はウェブブラウザが搭載する JavaScript エンジン上で動作しているため，ウェブブラウザが OpenCL 呼び出しに対応していない限り，JavaScript から OpenCL を利用することはできない．主要なブラウザのうち，このような OpenCL の呼び出しに何らかの対応を行っているものは，2011 年 4 月時点で存在しない．

そこで，動的リンクによって各種ブラウザに導入するプラグインを開発し，このプラグインを利用することで，ブラウザ上の JavaScript から OpenCL を呼び出せるようにした．

4.2　NPAPI

ブラウザ本体とプラグインをつなぐ API として，NPAPI を利用した．NPAPI (Netscape Plug-in API) は，プラグインとブラウザがやり取りを行うためのインタフェースを提供するフレームワークである．

NPAPI は，様々なメディアをブラウザ上で扱うための規格である．各プラグインは，application/x-foobar のような MIME タイプと関連付けてブラウザに登録される．ウェブページ内に埋め込まれたメディアの MIME タイプが登録された MIME タイプと一致していれば，対応するプラグインを呼び出して，そのプラグインにメディアの処理を行わせる．これにより，ブラウザが対応していない最新のメディアフォーマットが登場しても，プラグインを導入するだけで，ブラウザの対応を待たずにそのメディアを扱えるようになる．

NPAPI そのものは多様なメディアを扱うためのものであるが，その拡張として npruntime と呼ばれるものがある．npruntime は，埋め込まれたメディアを JavaScript オブジェクトとして取り扱い，操作可能にするものである．npruntime を利用すると，埋め込まれたメディアを JavaScript からのメソッド呼び出しなどによって操作できる．JavaScript からのメディアへの操作は，NPAPI (npruntime) を通じてプラグインに渡され，プラグインは JavaScript から渡された情報をもとに適切な処理を行う．プラグインは，近年のブラウザで広く普及している機能拡張手段であるアドオンや拡張機能方式とは異なり，ネイティブコードにビルドされる形で開発されるため，必要であれば外部ライブラリなどをプラグイン内で呼び出すこともできる．

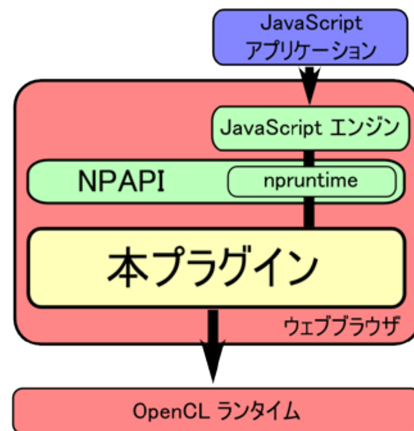

Fig.1. NPAPI を利用したプラグイン実装モデル

4.3　プラグインの構成

ウェブブラウザ上の JavaScript が OpenCL コードを呼び出すと，NPAPI からプラグインに対して，JavaScript が呼び出そうとしているメソッドの名称，引数の数，引数の型，引数データなどの情報が渡される．JavaScript の型や値とプラグインのコードを記述している C++言語の型や値の対応を Table1 に示した．

Table1. JavaScript の型と C++言語の型の対応

JavaScript の型（値）	C++言語の型（値）
null	NULL
undefined	void
boolean	bool (int)
number	int32 もしくは double
その他のオブジェクト	void *

情報を受け取ったプラグインは，メソッド名から OpenCL ランタイム API が提供する関数のうち，どれを呼び出すのが適切なのか検査する．また，引数の数とそれぞれの引数型およびそれぞれの引数が持つ情報を検査し，メソッド名と設定された引数が適切なのかを判定する．NPAPI から渡された情報の検査が終了し，すべてのデー

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.

タが適切であれば，必要な一時領域の確保や引数のマッピングなどの事前処理を行い，対応する OpenCL ランタイム API 関数を実際に呼び出す.

呼び出した OpenCL ランタイム API 関数が返ってくると，どのような返り値が返されたかをチェックし，返り値を JavaScript 用の適切な形式に変換し，NPAPI を通じて JavaScript に返す.

また，上述の処理を簡略化した処理により，定数呼び出しへの対応も行う.

4.4　プラグインの実装

プラグインの実装は，前節で述べた処理を各 OpenCL ランタイム API 関数へ対応させるものが主となる.

OpenCL ランタイム API が提供している関数は，バージョン 1.0 で 61 個ある．本実装では，この関数のうち基礎的で必須と考えられる 39 関数と，JavaScript の言語仕様などを考慮して改変あるいは新設した独自の 6 関数の計 45 関数の呼び出しに対応した．残り 20 個の関数については，必ずしも必要ではないあるいは JavaScript で利用するにあたっては意味をなさないなどの理由から実装していない.

プラグインは C++言語で実装した．プラグインサイズは，Microsoft Windows 向けの DLL 形式で 200KB ほどになる.

プラグインが利用可能なブラウザとしては，Mozilla Firefox や Google Chrome，Opera，Safari が挙げられる．ただし Microsoft Internet Explorer については，NPAPI に対応していないことから利用できない.

プラグインはウェブサイト（http://blog.tommy6.net/）で公開しており，自由にダウンロードして利用できる.

5.　アプリケーションの構築と実証実験

本システムを利用した簡単なアプリケーションを作成し，本システムによってどのようなことがウェブアプリケーションで実現可能になるのかを実証的に示す.

5.1　アプリケーションの概要

現在の多くのウェブサイトでは，HTML のフォーム機能や JavaScript の動的即時コード実行機能および非同期通信機能などを利用し，ページ遷移することなくユーザの様々な操作に応じた応答を提示できる．本システムのアプリケーションでも，ウェブフォームによる手軽な GPGPU コード投稿および即時実行モデルを提供することで，気軽に GPGPU プログラミングを体験できる環境の実現を考えた.

また，ソーシャルネットワークサービスに代表されるように，各ユーザ個人で完結するのではなく，他の多数のユーザとひとつのサイト上で情報を共有できるサービスが普及している．この形態を利用し，ユーザ同士で GPGPU コードを共有することで，各ユーザの GPGPU プログラミング技能の向上に寄与するプラットフォームの構築を考えた.

この 2 つをあわせ，サイトが提示した様々な問題を解く GPGPU コードを参加者が投稿し，互いに助言などを行うことによって GPGPU プログラミングの技能を高めあうオンライン学習サイトを構築した.

具体的には，サイトの参加者は，はじめにサイト上に提示された様々な問題の中から一つを選択し，その問題を解く回答コードを，ウェブフォームを通じてサーバに保存する．各ユーザが投稿した回答コードはサーバ上に蓄積され，適切なチェックを経た後実行可能コードとしてサイト上に提示される．ユーザは回答コードの中から任意の回答コードを選択し，それを自分の計算機環境上で実行させる．その結果，処理時間などのパフォーマンスデータが収集され，サーバ上に保存される．各回答コードに対応する様々なユーザ環境（ユーザが実際に利用したブラウザや CPU，GPU についての情報）におけるパフォーマンスデータがサイト上に提示されるので，各ユーザはこれを参考にコードの改良や再投稿を行う.

5.2　アプリケーションの利用の流れ

サイト上には本システムを利用して処理されることを想定した問題が一覧で表示される（Fig.2）．各ユーザは，提示された問題の中から自分が解きたい問題をひとつ選択する．問題を選択すると，選択した問題に対する回答コードを入力するためのウェブフォームが表示される.

Fig.2. サイト上に提示される問題のリスト

```
cl.clSetKernelArg(kernel_mul_vec4, 1, memB);
cl.clSetKernelArg(kernel_mul_vec4, 2, memC);
cl.clSetKernelArg(kernel_mul_vec4, 3, memMSize);

for(var i = 0; i < matrix_size; i++) {
        cl.clEnqueueWriteFloat32Array(command_queue, memA, cl.CL_SIZE_FLOAT32*i*matrix_size, m
        cl.clEnqueueWriteFloat32Array(command_queue, memB, cl.CL_SIZE_FLOAT32*i*matrix_size, m
}
cl.clEnqueueWriteInt32Array(command_queue, memMSize, 0, 1, [ matrix_size ]);

for(var i = 0; i < loop_count; i++) {
        ret = cl.clEnqueueNDRangeKernel(command_queue, kernel_mul_vec4, 1, null, [62500], [250
        if(ret != cl.CL_SUCCESS) {
                error_check = false;

                break;
        }
        cl.clWaitForEvents(1, [ my_event ]);
```

Fig.3. ユーザが投稿するコードの例

ユーザは選択した問題を処理するためのコード（Fig.3）をフォームに入力し，サーバに送信する．ユーザが送信した回答コードはサーバ内のデータベースに，対応する問題と紐付けられて蓄積される.

サーバに蓄積された回答コードは，各問題とそれに対応する回答コードとして，一覧でユーザに提示される．ユーザは，様々な回答コードの中から任意のものを選択し，自分自身の計算機環境でそのまま即時実行することができる．すなわち，選択した回答コードはユーザの Web ブラウザが現在動作している計算機上の CPU や GPU などの OpenCL デバイス上で実行され，処理に要した時間

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.

などのパフォーマンスデータが収集される．収集されたパフォーマンスデータは，回答コードの識別子や実行デバイスのデータとともにデータベースに保存され，パフォーマンス統計がユーザに示される（Fig.4）．

Fig.4. 収集したパフォーマンスデータ提示の一例

回答コードを投稿したユーザは，提示された統計データを他のユーザと比較することにより，自身のコードのパフォーマンス上の問題を評価することができる．

なお，本アプリケーションの構築にあたっては，ユーザの入力や操作に応じた動的ページ生成サーバを PHP で実装し，データベースとして MySQL を用いた．

5.3 実証実験

GPGPU コードの違いによってどれだけのパフォーマンス差が生じるかを実際に確かめるため，アプリケーション上の問題例および回答コード例を設定し，実行時間を計測する実証実験を行った．

5.3.1 実験設定

ウェブサイトに提示される問題として，500 次の正方行列 A の A^{10} を計算する問題を設定した．この問題に対する回答例として以下の 3 つの方法を設定し，それぞれについての処理時間を取得する．なお本実験で対象とするのは実際の計算時間のみであり，メモリコピー等の時間は対象としない．

1. JavaScript のみで処理を行う方法（比較用）．
2. 本システムを利用し，シングルスレッドで CPU および GPU に処理を行わせる方法．
3. 本システムを利用し，問題を分割した並列処理コードによって，CPU および GPU に処理を行わせる方法．

5.3.2 実験環境

本実験で使用した環境は Table2 のとおりである．

Table2. 実験環境

ブラウザ	Google Chrome + 本プラグイン
CPU	Intel Xeon E5520 2.27 GHz (4 Cores/8 Threads)
RAM	DDR3-1333 ECC 8GB
GPU	AMD Radeon HD 5750 512MB

5.3.3 実験結果

回答 1 で処理した結果を Table3 に示す．また，JavaScript のみによる処理を，本システムを用いてシングルスレッドのまま CPU をおよび GPU に処理させた結果を Table4 に示す．

Table3. JavaScript のみによる処理に要した時間

実行時間 [ミリ秒]	148,000

Table4. 本システムを利用したシングルスレッドによる処理に要した時間

デバイス	CPU	GPU
実行時間 [ミリ秒]	4,870	2,000,000（推定値）

本システムを利用し CPU で処理を行わせると，回答 1 の JavaScript のみによる処理と比べて 30 倍程度の高速化が得られることがわかる．一方で，GPU に処理を行わせると，逆に大幅に処理時間を要している．これは，JavaScript による処理をそのまま移植しただけのシングルスレッド処理であり，GPU の多並列構造を活用しきれていないためである．

回答 3 では，問題を分割して得られた処理ブロックを OpenCL の並列実行モデルを利用して並列処理させる．様々な分割度による実行時間の結果を Table5 に示す．

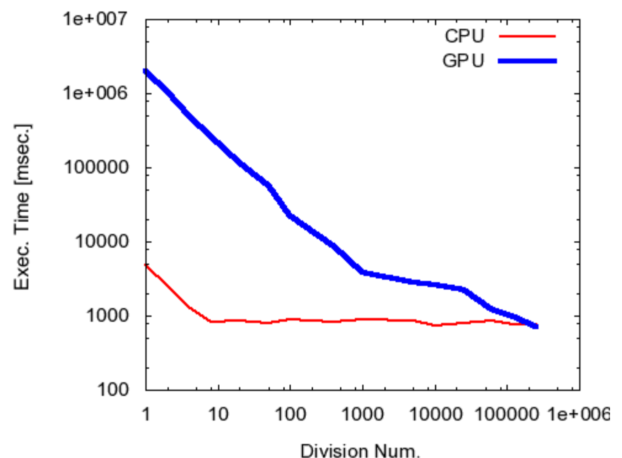

Fig.5. 本システム利用した並列処理に要した時間

問題を分割して並列処理を行うと，CPU と GPU ともに処理時間が短縮される．しかし，CPU と GPU にはパフォーマンス向上の特徴に大きく差があることがわかる．

現行の普及版の CPU は 2〜6 コア程度であるため，並列度を大きくしてもパフォーマンスの向上はコア数とほぼ同程度にとどまり，数十以上の分割による効果は低い．これに対し，GPU は分割数を上げれば上げるほどパフォーマンスが向上する．GPU のアーキテクチャが数十万スレッドにも至る超並列処理を想定しており，むしろ十分な並列化を行わないと CPU よりも劣るパフォーマンスしか得られない．このような分割数のパフォーマンスに及ぼす効果をユーザのブラウザ環境を用いて容易に体験することができる．

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.

6. まとめと今後の課題

本論文では，ブラウザ上の JavaScript から OpenCL を呼び出すことを可能にするウェブブラウザプラグインの開発について述べた．また，このプラグインを用いた簡単なアプリケーションを構築し，このプラグインの利用によって可能になるウェブアプリケーションでの実証実験について述べた．

プラグインを利用するブラウザによっては，メモリデータのコピーなど一部の操作において著しくパフォーマンスが低下する現象を確認しており，これらの原因究明と解決が今後の課題である．

参考文献

[1] Version 1.0, The OpenCL Specification
http://www.khronos.org/registry/cl/specs/opencl-1.0.pdf
[2] Version 1.1, The OpenCL Specification
http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf
[3] Gecko Plugin API Reference - MDC Doc Center
https://developer.mozilla.org/en/Gecko_Plugin_API_Reference
[4] OpenCL 入門- マルチコア CPU・GPU のための並列プログラミング，株式会社フィックスターズ，2010
[5] Programming Massively Parallel Processors , David B. Kirk, Wen-mei W. Hwu, MORGAN KAUFMANN, 2010.

Copyright © 2011 by Information Processing Society of Japan and
The Instiute of Electronics, Information and Communication Engineers
All rights reserved.